Decentralized Multi-Agent Control from Local LTL Specifications

Ioannis Filippidis, Dimos V. Dimarogonas and Kostas J. Kyriakopoulos

Abstract—We propose a methodology for decentralized multi-agent control from Linear Temporal Logic (LTL) specifications. Each agent receives an independent specification to formally synthesize its own hybrid controller. Mutual satisfiability is not a priori guaranteed. Due to limited communication, the agents utilize meeting events to exchange their controller automata and verify satisfiability through model checking. Local interaction only when common atomic propositions exist reduces the overall computational cost, facilitating scalability. Provably correct collision avoidance and convergence is ensured by Decentralized Multi-Agent Navigation Functions.

I. INTRODUCTION

There have been multiple approaches to multi-agent control. Both classic motion planning [1], [2], [3] and task related methods [4], [5] have been developed. The current effort targets unification [6], [7], [8], [9], [10], [11], [12], [13]. Since we currently deal with increasingly complex and heterogeneous systems, decentralization is desired for scalability, while safe and guaranteed results are required. Formal methods for specification and automatic synthesis of provably correct controllers can ensure this. The system’s specification can be provided in a logic sufficiently expressive for the desired tasks.

In [13] centralized multi-agent systems with perfect information are considered. Synthesis of a centralized controller is performed from a global LTL specification. This requires a globally connected multi-agent system to ensure information availability. Necessary and sufficient conditions for a global specification to be decomposable to bisimilar local ones are derived in [11]. This provides a distributed method from top to bottom. Execution with no communication limitations is assumed. In [14] the issue of communication and synchronization is analyzed and a solution is proposed, which diagnoses whether an LTL specification needs communication or not. Similarly to [15], where only communicating agents are allowed to move, this check characterizes the subset of realizable specifications.

We extend here the application of formal methods to decentralized multi-agent systems. The method proposed enables each agent to independently synthesize safe controllers, verify its plans versus those of others upon meeting them and execute them in a continuous state space using Navigation Functions. The main innovations with respect to previous works are that tasks are defined independently in a bottom up manner, there is no global specification for all agents, the verification is decentralized and the motion planning controllers are NFs.

In particular, LTL specifications provided to the agents are not produced in a centralized way, hence they may be contradicting each other. The solution proposed for this aims at gradually verifying that agent specifications are mutually satisfiable. Events of path-connectedness enable exchange of their alphabets and automata, to allow model checking [16]. Moreover, even if mutually satisfiable, we are interested in cases in which long-range communication is not available. If path-connectivity is absent when required by agents, the controllers will fail to act according to their specifications, due to lack of information. We embed in LTL communication requests when information is needed and implement them using additional follower agents under connectivity maintenance control.

For interfacing the discrete controllers to the continuous system state we choose Navigation Functions (NFs) [17], [2]. NFs are continuous feedback motion planning control laws [3] which ensure collision avoidance and convergence. As a result, the specification is formally satisfied in the discrete control level, which in turn is interfaced to the continuous state space via provably correct NF controllers.

The rest of this paper is organized as follows: preliminaries are covered in § II, the problem defined in § III, the layered architecture is described in § IV, decentralized verification in § V, and the method illustrated by simulation in § VI. Concluding remarks are summarized in § VII where future research is considered.

II. PRELIMINARIES

A. Linear Temporal Logic

LTL is an extension of propositional logic suitable for reasoning about infinite sequences of states [18]. Let $P$ be a set of Atomic Propositions (APs) [16]. More complex formulae result from combining propositional ($\neg, \land, \lor, \to, \leftrightarrow$, i.e., negation, conjunction, disjunction, implication, equivalence) with temporal ($\mathcal{U}$, $\Box$ and $\diamond$, i.e., until, always, eventually) operators. Formula $\phi_1 \mathcal{U} \phi_2$ requires that $\phi_1$ be true until $\phi_2$ becomes true, which is required to happen. Formula $\phi_1 \rightarrow \phi_2$ requires that $\phi_1$ be true at some future time. Its dual $\phi_1 \rightarrow \phi_2$ requires that $\phi_1$ be true at all future times.

The semantics of LTL are defined with respect to sequences $\sigma : N \to 2^P$. A formula is evaluated over $\sigma$ by starting its interpretation from $\sigma(0)$. Let $\Phi_P$ denote the set of
well-formed formulas [19] over \( P, p \in P \) and \( \phi_1, \phi_2 \in \Phi_P \).

By \( \sigma \models \phi \) we mean that \( \phi \) is true when evaluated over \( \sigma \).

- For all \( \sigma \) it is \( \sigma \models \phi \) true and \( \sigma \not\models \phi \) false;
- \( \sigma \models p \) if and only if (iff) \( p \in \sigma(0) \);
- \( \sigma \models \neg p \) iff \( p \notin \sigma(0) \);
- \( \sigma \models \phi_1 \land \phi_2 \) iff \( \sigma \models \phi_1 \) and \( \sigma \models \phi_2 \);
- \( \sigma \models \phi_1 \lor \phi_2 \) iff \( \sigma \models \phi_1 \) or \( \sigma \models \phi_2 \);
- \( \sigma \models \phi_1 \implies \phi_2 \) iff \( \exists i \in \mathbb{N} : \sigma^i \models \phi_2 \) and \( \sigma^j \models \phi_1 \) for all \( j < i \), where \( \sigma^m(k) \triangleq \sigma(m + k) \) for all \( k \in \mathbb{N} \).

**B. Navigation Functions**

Let \( x_i \in \mathbb{R}^n, i \in I \subset \mathbb{N} \) be the continuous states of a set of agents indexed by \( I \). Navigation Functions (NF) are potential fields free of local minima introduced in [17]. NFs were extended to decentralized multi-agent systems in [2]. Connectivity maintenance constraints for NFs were proposed in [20]. Each agent is holonomic, \( \dot{x}_i(t) = u_i(t) \). The control input \( u_i \) is chosen as \( u_i \triangleq -L_i \varphi_i(x(t), x_i) \) where \( \varphi_i \triangleq (\dot{x}_i - \dot{x}_i)^T \) is the NF of agent \( i \) and \( x(t) \) is the stack vector of all \( x_i \). Function \( \varphi_i \) is maximal in collision sets and has a unique minimum at \( x_i \).

Let \( N_i \) be the set of neighbors to which agent \( i \) should remain connected. The destination function \( \gamma_i \triangleq \sum_{j \in N_i} \| x_i - x_j \|^2 \) is used later for followers and \( \gamma_i \triangleq \| x_i - x_{ij} \|^2 \) for leaders, described in § III-A. The destination \( x_{ij} \) for a leader is selected by the discrete control layer. Function \( G_i \) measures the proximity between agents, in order to avoid collisions. Also, if connectivity between two agents is maintained, then \( G_i \) prevents them from separating. For its definition, please refer to [20], [2].

In order for \( \varphi_i \) to acquire the properties of a NF, the parameters \( k, \lambda \) and \( h \) should be larger than thresholds which depend on each problem, as proved in [20]. Leaders constrained by connectivity can reach their destinations when sufficiently many followers are available to connect them over that distance.

**III. PROBLEM DEFINITION**

**A. Agent Continuous and Discrete States**

We consider a set of \( N \in \mathbb{N}^* \triangleq \mathbb{N} \setminus \{0\} \) agents, indexed by \( I_0 \triangleq [1, \ldots, N] \). Each agent \( a_i \) is characterized by a continuous state \( x_i \in X_i \subseteq \mathbb{R}^n \) where \( n_i \in \mathbb{N}^* \). In this work the continuous state is the position of each agent on the Euclidean plane, \( X_i = \mathbb{R}^2 \). In addition, it may have a discrete state \( q_i \in Q_i \), associated with it, where \( Q_i \subseteq \mathbb{N}^m \) and \( m_i \in \mathbb{N} \). For example, an agent may carry a red beacon described by the discrete state \( q_i \). The beacon can be on or off, so \( q_i \in Q_i = \{0, 1\} \). The combination of continuous and discrete state forms a hybrid state \( H_i = x_i \times q_i \).

Each agent is either a leader \( l_i \) or a follower \( f_i \). Each leader receives its own LTL specification \( \varphi_i \). Followers do not receive specifications. There are \( N_l \) leaders indexed by \( I_l \subseteq I_0 \) and \( N_f \) followers indexed by \( I_f \triangleq I_0 \setminus I_l \).

Define the ball \( B_{\rho_i}(x_i) \triangleq \{ x \in \mathbb{R}^n \mid \| x - x_0 \| < \delta \} \) with center \( x_i \) and radius \( \delta > 0 \). We assume that each agent occupies \( B_{\rho_i}(x_i) \), where \( \rho_i > 0 \).

If \( x_j \in B_{R_s}(x_i) \), then agents \( a_i \) and \( a_j \) can communicate directly, Fig. 3a. We call \( R_s \) the communication radius. Agents \( a_i \) and \( a_j \) are path-connected if there exists a chain connecting them, comprised of pairs of directly connected agents, Fig. 3b. This chain relays information between \( a_i \) and \( a_j \), so they can communicate.

**B. Atomic Propositions observing the Continuous State**

We are now going to discuss the problem’s modeling. We want to specify the movement of leaders between points using LTL. To “speak” in terms of “point” entities, the proximity to each point \( y \) is expressed by an AP associated with \( y \). Let \( p^o_{ij} \) be the \( j \)th observer AP of agent \( a_i \), observing the state of agent \( a_k \), then

\[
p^o_{ij} \triangleq \begin{cases} 
\text{true, } \| x_k - y \| < \varepsilon & \text{if } i, k \in I_l, \ y \in X_k \\
\text{false, } \| x_k - y \| \geq \varepsilon & \text{otherwise}
\end{cases}
\]

where \( \varepsilon > 0 \) is some constant such that \( \varepsilon < R_s \) and \( y \) is some point of interest to \( a_i \). For sufficiently small \( \varepsilon \) at most one observer can be true at any time \( t \). Let \( P^o_i \triangleq \{ p^o_{ij} \}_{j \in I_i} \) be the observer APs of agent \( a_i \), where \( I_i^o \triangleq \mathbb{N} \setminus n_i^o \) and \( n_i^o \in \mathbb{N} \).

Alternatively, an observer may be defined to measure the distance between the continuous states of two different agents \( a_i \) and \( a_k \), with \( i \neq k \):

\[
p^q_{ij} \triangleq \begin{cases} 
\text{true, } \| x_i - x_j \| < \varepsilon & \text{if } i \neq j \\
\text{false, } \| x_i - x_j \| \geq \varepsilon & \text{otherwise}
\end{cases}
\]

The value of \( p^q_{ij} \) cannot be instantly controlled by agent \( a_i \). However, if \( p^q_{ij} \) depends on \( x_i \), then \( a_i \) can change it at some later time by using a continuous controller.

Each observation \( p^o_i \) on the state of some other agent \( a_k \), \( k \neq i \) introduces a dependency of \( a_i \) on the behavior of \( a_k \). Consider the directed graph defined by the dependencies. Only the subgraph which is reachable from \( a_i \) can affect it and needs to be modeled by \( a_i \) during verification in § V.

**C. Atomic Propositions controlling the Continuous State**

The continuous movement of each agent is controlled by a decentralized Navigation Function, defined in § II-B. The destination used in the NF can change in time, according to the specification \( \varphi_i \). Let \( x_{dij} \) be the destinations used by agent \( a_i \), where \( j \in I_i^c \triangleq \mathbb{N} \setminus n_i^c \) and \( n_i^c \in \mathbb{N} \). We will refer to each NF \( u_{ij} = -\nabla_{x_i} \varphi_i(x,x_{dij}) \) as a different continuous controller, associated with \( x_{dij} \). Define the continuous controller \( AP \) \( p^c_{ij} \) to be True when \( u_{ij} \) is active, False otherwise. Each NF is asymptotically stable from almost all initial conditions (apart from a subset of measure zero). So it reaches \( B_{\varepsilon}(x_{dij}) \) in finite time, which motivated the definition of \( p^c_{ij} \). The value of \( p^c_{ij} \) is controllable only by agent \( a_i \) and no other. At most one NF is active at any time, expressed as \( \square \neg (p^c_{ij} \land p^c_{ik}) \) for all \( j \neq k \). The discrete control layer selects which \( p^c_{ij} \) to activate, as described in § IV-A.

Note that by defining \( x_{dij} = x_k + c_{dij} \) for some relative position \( c_{dij} \in \mathbb{R}^2 \) with respect to some other agent \( a_j \), formation control can also be achieved. Let \( I_i^c \triangleq \{ p^c_{ij} \}_{j \in I_i^c} \).
D. Atomic Propositions observing Path-Connectedness

If \( a_i \) cannot communicate with \( a_k \), then the value of each \( p^c_{ij} \) which depends on \( x_k \) is unobservable. In this case, the discrete control layer designed later cannot decide what action to take. Define the AP \( p^w_{ik} \) to be True when \( a_i \) and \( a_k \) are path-connected, False otherwise.

If \( p^w_{ik} = \text{false} \) implies that the value of \( \phi_i \) is independent of all \( p^c_{ij} \) which depend on \( x_k \), then limited communication cannot result in undecidable situations for the discrete controller. So tasks which require communication can “wait”.

Let \( \phi^i \) be some task which requires communication of \( a_i \) with \( a_k \). Then, if \( \phi_i \) requires that eventually \( \phi^i \) be satisfied, it assumes a priori that \( a_i \) will communicate with \( a_k \) in the future. So \( \phi^i \) should either include the assume-guarantee subformula \( \diamond p^c_{ij} \rightarrow \phi^i \), or the assumption about eventual path-connectedness be modeled during verification. Let \( P_i^w \triangleq \{ p^w_{ij} \} \in I_w \) and \( I_w \subseteq I_a \setminus \{i\} \).

E. Atomic Propositions maintaining Connectivity

As already discussed, certain subformulas of \( \phi_i \) may require information about other agents. So \( a_i \) needs a means to remain connected with selected other agents as shown in Fig. 3. This is achieved by using the connectivity maintenance algorithm from [21]. We define the APs \( p^m_{ij} \) which control the connectivity. Agent \( a_i \) can request from the network to maintain its connection with \( a_k \) by setting \( p^m_{ik} \) to True, or disconnect by setting it to False. Note that disconnection requires that both agents \( a_i \) and \( a_k \) broadcast a disconnection request. Connections are requested only when path-connected, specified as \( \Box(\neg p^m_{ik} \lor p^m_{ki}) \), and connectivity maintenance is modeled by \( \Box((p^m_{ik} \land p^m_{ki}) \rightarrow \Box p^m_{ik}) \). Let \( P^m_i \triangleq \{ p^m_{ij} \} \in I_w \).

F. Problem Statement

Let \( P_i \triangleq P^c_i \cup P^m_i \cup P^w_i \cup P^o_i \) be all the APs of leader \( l_i \). Our aim is to make the agent visit the points of its APs and connect with other agents as specified by \( \varphi_i \).

Each leader receives an LTL specification \( \varphi_i \in \Phi_{P_i} \). We are interested in each agent independently synthesizing its own hybrid controller, such that it will satisfy safety properties specified by \( \varphi_i \) and check liveness properties when exchanging information with other agents.

IV. LAYERED ARCHITECTURE

Each agent communicates with others, maintains connectivity with selected neighbors as needed, takes decisions on a discrete level and moves in the continuous state space. So its controller is hybrid, Fig. 1c, comprising of a discrete and a continuous control layer, Figs. 1a and 1b. These are augmented by a layer controlling connectivity and the communication layer, Fig. 2a. We describe this layered architecture briefly here, while more details can be found in [22].
be used as the discrete control layer. The produced strategy is guaranteed to be safe, but liveness is not guaranteed. This approximates the partial solution of an adversarial game.

B. Continuous Control Layer

The continuous control layer is responsible for implementing the NF $\Phi_\ell(x, x_{di}, \lambda)$ which is associated with the currently active $p_{ij}^\ell$ selected by the discrete control layer. So each NF executes a transition between points, when required by $\phi_i$, as shown in Fig. 1b. Each transition requires some execution time, during which the automaton executes and evaluates observation propositions.

C. Communication Layer

When agents $a_i$ and $a_j$ become path-connected, they can communicate. For the purpose of deciding if there is no logical conflict between their specifications, the two agents should exchange information about their behavior. The behavior of each agent comprises of its discrete control layer. This is defined by the set $P_i$ of AP and the controller automaton $A_i$. Note that the AP in $P_i$ define the continuous and connectivity control layers. So, each agent transmits $\{P_i, A_i\}$ when meeting another agent.

Some observers $p_{ij}^\ell$ of $a_i$ cannot be “eventually” controlled by its NFs, because their value depends on the state $x_k$ of another agent. So $a_i$ does not control $x_k$, but is affected by it. For this reason it needs information about their behavior. Furthermore, connectivity maintenance requests are broadcast and received through the communication layer.

D. Connectivity Maintenance Layer

If agents $a_i$ and $a_j$ are not path-connected, then $a_i$ cannot observe state $x_j$. Observables of $a_i$ depending on $x_j$ cannot be evaluated in this case. So the specification of $a_i$ should be independent of $x_j$, when $x_j$ is unobservable.

However, certain tasks may require maintenance of connectivity. The role of followers is to function as communication relays that re-transmit information between leaders which have requested to remain connected. We use the distributed connectivity maintenance algorithm from [21]. The connectedness criterion has been adapted here to maintain the connectedness between only those leaders requesting it.

A sufficient number of followers is needed to allow the leaders to stretch as far away from each other as their specifications require. A conservative estimate of the required number of followers per pair of leaders can be obtained by finding the largest distance between points over which propositions in $P_i$ and $P_j$ are defined and divide it by $R_s$.

V. DECENTRALIZED VERIFICATION

A. Logical conflicts between specifications

As detailed in § III, the agents receive independent specifications in a decentralized manner. Therefore, it is not guaranteed that $\Phi_i$ and $\Phi_j$ will agree, they may be logically inconsistent with each other.

For example $\Phi_i$ may tell $a_i$ to wait at point $A$ until $a_j$ appears, then go to point $B$, as shown in Fig. 4. Assume that $a_j$ starts at point $C$. If $\Phi_j$ tells $a_j$ to remain at $C$ forever, then it will never appear at $A$. So $a_i$ will never start going to $B$ and $\Phi_i$ cannot be satisfied, Fig. 5.

This result is due to the logical conflict between the specifications of the two agents. It becomes apparent that in order for all agents to be able to satisfy their specifications, their specifications should in some way agree with each other. Rich specifications are still possible (e.g. $a_j$ visiting another 10 points before $A$). However, logical compatibility of $\Phi_i$ and $\Phi_j$ is not ensured a priori. For this reason, it is necessary for the agents to decide whether there is a logical conflict between their specifications while executing them.

This elicits the question of what happens in case the specifications are not mutually satisfiable, due to logical conflicts. In that case it is necessary that some of the specifications be changed, e.g. relaxed in some way. This procedure will be a refinement of one, or both, of the specifications, so that they become logically consistent. This reconfigurability of the system constitutes an interesting direction of future research, related to [26].
Fig. 6: In order to identify common atomic propositions, communicating agents should interpret them similarly. For simplicity $P^o_i$, $P^w_i$ are not shown.

B. Verification Procedure

After appropriately modeling agent behaviors, all possible executions of the system can be examined using model checking. Here we describe how the system is modeled in the Promela programming language, in order to be checked with the SPIN model checking software [27]. More details about the modeling and the custom interface between MATLAB and SPIN can be found in [22].

Only those agents on which $a_i$ depends through its observables need to be modeled by $a_i$. As $a_i$ meets more agents, it can model its environment better.

The issue of state explosion imposes a limitation to the number of agents modeled. This is partially mitigated by the relatively small number of points interesting each agent (as opposed to grids) and the dependencies between agents, which are assumed to be sparse, as will usually be the case in a scenario involving distributed robotic teams.

1) Alphabet Comparison: If an agent $a_j$ has an NF controller $p^c_{ij}$, then it also has an associated observer $p^o_{ij}$. If agent $a_i$ observes with $p^o_{ij}$ the same point as $p^c_{ij}$, then these two observers are identical. In order to find which observer APs are common, Fig. 6, the alphabets $P^o_i$ and $P^w_i$ are compared using their definitions. In more detail, the definitions of $p^c_{ik}$ and $p^o_{jr}$ comprise the indices of the agents on whose states each AP depends (assuming both agents use the same index set $I_a$), together with the points of interest. By comparing the indices and points, observers of different agents can be compared. Note that the controllers $p^c_{ij}$ of different agents cannot be the same. The APs in the collected alphabets are initialized to false for the verification.

2) Modeling agent discrete controllers: The controller automaton constructed in § IV-A is modeled as an independently executing deterministic process, which selects the values of controllable $p^c_{ij}$ and $p^c_{ij}$ based on the current values of $p^o_{ij}$ and $p^o_{ij}$.

3) Modeling Navigation Function Controllers: NFs guarantee transitions between points in finite time (a priori unknown), abstracting them to the discrete level. This is modeled by introducing each NF controller as a separate process, which monitors when $p^c_{ij}$ becomes true, so then it sets $p^o_{ij} = true$. Fairness is enforced during model checking, to ensure that the NF controller will eventually execute after a non-deterministic number of iterations, allowing any possible interleaving to be checked.

4) Modeling agents unknown yet: As already mentioned, the agent has initially limited knowledge about its environment. In other words, it does not know what the other agents will do. Although, the verification process can be iteratively executed, we need some way of modeling the state of incomplete information about other agents, on which the one performing the verification, let that be $a_i$, depends.

This can be achieved by modeling all possible executions (since we do not know what is the particular behavior of the other agent). An independently executing process which changes the associated observable values between true and false in a non-deterministic way models this situation.

VI. SIMULATION RESULTS

A case study using the proposed algorithm involving $n_l = 6$ leader agents and $n_f = 3$ followers, illustrated in Fig. 7, in which agents $a_1, a_2$ (blue,green) should eventually patrol the lower-right area, visiting infinitely often two points one after the other. Agents $a_5, a_6$ (magenta,yellow) wait for $a_4$ (cyan) before requesting connectivity and moving to $x_{d51}, x_{d61}$. Agent $a_4$ goes first to $x_{d41}$, then to $x_{d42}$. Finally, $a_3$ (red) goes to $x_{d31}$, waits to see $a_2$ and then moves to $x_{d32}$.

Followers $f_7, f_8, f_9$ are available to provide path connectivity where needed.

The specifications are defined as

$$\phi_1 = \square \neg (p^1_{11} \land p^2_{12}) \land (p^2_{12} \rightarrow (p^1_{11} \land p^2_{12}))$$

$$\phi_2 = \square \neg (p^1_{21} \land p^2_{22}) \land (p^2_{22} \rightarrow (p^1_{21} \land p^2_{22}))$$

$$\phi_3 = \square \neg (p^1_{31} \land p^2_{32}) \land (p^1_{31} \land p^2_{32})$$

$$\phi_4 = \square \neg (p^1_{41} \land p^2_{42}) \land (p^1_{41} \land p^2_{42})$$

$$\phi_5 = \square \neg (p^1_{52} \land p^2_{53}) \land (\square (p^1_{52} \land p^2_{53}) \rightarrow \bigcirc p_{53}) \land (\square (p^1_{53} \land p^2_{52}) \land (\bigcirc p_{52} \land p_{52} \land (\bigcirc p_{51} \land p_{51}))$$

$$\phi_6 = \square \neg (p^1_{62} \land p^2_{63}) \land (\square (p^1_{62} \land p^2_{63}) \rightarrow \bigcirc p_{63}) \land (\square (p^1_{63} \land p^2_{62} \land (\bigcirc p_{62} \land p_{62} \land (\bigcirc p_{61} \land p_{61})$$

and the followers constantly execute a NF with neighbor list. The NF controller destinations are $x_{d11} = [0,0]^T$, $x_{d12} = [2, -2]^T$, $x_{d21} = [0, -1]^T$, $x_{d22} = [1,1]^T$, $x_{d31} = [-1, +1]^T$, $x_{d32} = [3, -1]^T$, $x_{d41} = [0,3]^T$, $x_{d42} = [-2, 1]^T$, $x_{d51} = [-3, 5]^T$, $x_{d61} = [2,4]^T$ and when $p^o_{51}, p^o_{62}$ are active, they issue connectivity requests to link $a_5, a_6$.

Let the observable APs be defined as follows

$$p^1_{11} = \langle x_1 - x_{d11} \rangle < 0.1$$

$$p^2_{11} = \langle |x_1 - x_{d12}| < 0.1, p^2_{12} = \langle |x_1 - x_{d12}| < 0.1, p^2_{12} = \langle |x_2 - x_{d22}| < 0.1, p^3_{11} = \langle |x_2 - x_{d11}| < 0.1, p^3_{12} = \langle |x_2 - x_{d12}| < 0.1, p^3_{21} = \langle |x_2 - x_{d21}| < 0.1, p^3_{22} = \langle |x_2 - x_{d22}| < 0.1, p^3_{31} = \langle |x_2 - x_{d31}| < 0.1, p^3_{32} = \langle |x_2 - x_{d32}| < 0.1, p^3_{41} = \langle |x_2 - x_{d41}| < 0.1, p^3_{42} = \langle |x_2 - x_{d42}| < 0.1, p^3_{51} = \langle |x_2 - x_{d51}| < 0.1, p^3_{52} = \langle |x_2 - x_{d52}| < 0.1, p^3_{61} = \langle |x_2 - x_{d61}| < 0.1, p^3_{62} = \langle |x_2 - x_{d62}| < 0.1, p^3_{71} = \langle |x_2 - x_{d71}| < 0.1, p^3_{72} = \langle |x_2 - x_{d72}| < 0.1, p^3_{81} = \langle |x_2 - x_{d81}| < 0.1, p^3_{82} = \langle |x_2 - x_{d82}| < 0.1, p^3_{91} = \langle |x_2 - x_{d91}| < 0.1, p^3_{92} = \langle |x_2 - x_{d92}| < 0.1$$

and $p^w_{51}, p^w_{62}$ detect path-connectivity between $a_5, a_6$ through followers. APs $p^o_{54}, p^o_{64}$ detect path-connectivity between $a_4$ and $a_5, a_6$, respectively, through any agent and function as information availability switches. Note that $p^o_{51}$ requires information about $x_2$, but it also functions as an information availability switch, because $|x_2 - x_3| < 1 < R_s$, so no additional observable is needed.
In the simulation $a_1, \ldots, a_4$ proceed to their objectives avoiding collisions, so that at $t_7$ agents $a_1, a_2$ have started patrolling the lower left area and $a_3$ is heading towards $x_{d32}$. At $t_8$ agent $a_4$ comes within distance 0.4 of $x_{d41}$, so that connectivity is triggered (thick continuous lines) and $a_5, a_6$ begin moving to $x_{d51}, x_{d61}$ respectively, while $f_7, f_8, f_9$ maintain the path-connectivity between $a_5, a_6$, as requested by $p_5^{m}, p_6^{m}$. Note that $a_5$ will conclude that $\phi_5$ is not realizable, until it learns the behavior of $a_4$ and models the assumption of eventual path-connectivity with $a_6$.

VII. CONCLUSIONS AND FUTURE WORK

An algorithm for converting independent LTL specifications to agent discrete controller automata manipulating continuous Navigation Function controllers has been introduced. Execution is decentralized and under limited communication. Mutual satisfiability is verified when agents meet and can exchange their automata. Further directions of research concern the introduction of reconfigurability capabilities, when agent specifications are conflicting.

REFERENCES